



The NSS HLA - Integration Framework



Integrated Training Program

Defense Modeling & Simulation Office
(703) 998-0660 **Fax (703) 998-0667**
hla@msis.dmsomil
<http://www.dmsomil/>

HLA Compliance

- **Adapting DoD simulations to HLA requires**
 - A plan on how to cleanly integrate RTI services *while preserving standalone maintenance and operation*
 - A strategy for managing message processing and simulated time
 - Significant software modifications and lots of testing
- **Characteristics of an ideal integration approach**
 - *Automate* sending/receiving updated *attributes*
 - *Simplify* initialization and sending/receiving interaction *events*
 - *Coordinate* federate's event processing with global federation time
- **Common framework would help tremendously**
 - Would support other DoD simulations requiring HLA compliance
 - Would speed up integration process - smaller learning curve
 - Would automate reliable and efficient book-keeping mechanisms

Outline of Presentation

- **Background**
 - **The Naval Simulation System (NSS)**
 - **The Joint Training Federation Prototype (JTFp)**
- **The NSS HLA-Integration Framework**
 - **Establishing the Thread of Control**
 - **Simplifying Object Declaration and Management**
 - **Automating Attribute Updates**
 - **Encapsulating Received Events into Objects**
 - **Coordinating Time Management**

The Naval Simulation System (NSS)

- **Discrete-Event Simulation on HP Workstations**
 - Platform-level modeling fidelity
 - Object-oriented (C++)
- **Used by CNO N812, CINCPACFLT, and Metron analysts**
 - Analysis, Operations planning, Training, Acquisition
 - *HLA integration is not allowed to impact standalone usage of NSS*
- **Software development environment**
 - 200,000 lines of C++ code distributed across more than 2,000 files
 - Team of up to 10 software developers and 7 analysts
 - *Strict configuration control with nightly regression testing*

The Joint Training Federation prototype (JTFp)

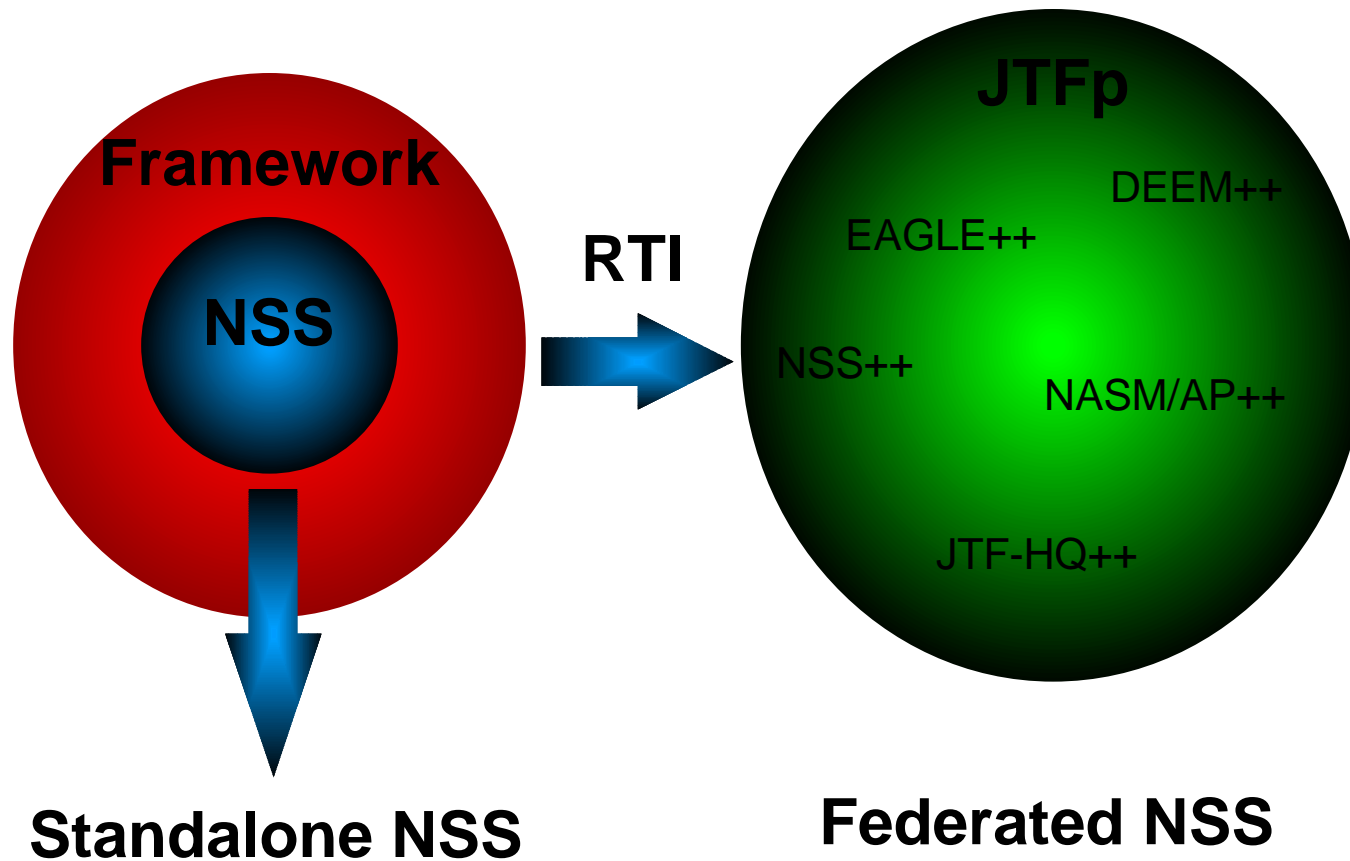
- **Characteristics of JTFp**
 - Real-time and logical-time modes of operation
 - Ownership management services exercised
 - Theater missile defense & marine landing scenarios
 - Integration in Huntsville, Alabama at AEgis Research

| Name | Model | Organization | Language | Time Management |
|---------|-------------|--------------|-----------|--------------------|
| EAGLE | Ground | MITRE | LISP | Time step |
| NASM/AP | Air Force | CACI | MODSIM II | Process model |
| NSS | Navy | Metron | C++ | Discrete-event |
| DEEM | Environment | ANL | SmallTalk | Multiple time step |
| JTF-HQ | Com. Ctrl. | AEgis | C++ | Wall clock |

The NSS Four-Step Integration Strategy

- **Develop the Framework in standalone mode**
 - Establish that the framework supports RTI services
 - Help debug the prototype RTI
- **Integrate NSS with the Framework**
 - Federate NSS with NSS
 - Verify NSS can communicate with the RTI through the Framework
- **Integrate the Framework with JTFp**
 - Coordinate initialization procedures and basic connectivity
 - Verify message formats
- **Integrate NSS and the Framework with JTFp**
 - Test and debug remotely
 - Maintain strict configuration management of NSS

Standalone and Federated NSS



Thread of Control

- **Framework provides the main program**
 - **Coordinates advancement of time**
 - **Decides when it is safe for federate to process next event**
 - **Handles messages passed between federate and the RTI**
 - **Manages book-keeping of object and attribute IDs**
- **Federate provides five subroutines**
 - ***void HLA_PubSub()***
 - ***void HLA_Instantiate()***
 - ***double HLA_ProcessUpTo(double time)***
 - ***double HLA_SimTime()***
 - ***void HLA_PostEvent(PostEventObject *PEO)***

Publishing and Subscribing

- **Framework provides**
 - Single line function calls for publishing and subscribing
 - Flexible argument list that accommodates any number of attributes
 - String-based interface that hides the RTI's data structure
 - Automatic exception handling
 - Automatic book-keeping of attribute and object IDs
- **Federate provides**
 - Interaction name, object class name and its associated attributes
 - Examples:
 - ♦ *HLARTI_publish("Ship", "Lat", "Lon", "Speed", NULL);*
 - ♦ *HLARTI_subscribe("Aircraft", "Lat", "Lon", "Alt", "Speed", NULL);*
 - ♦ *HLARTI_publish_interaction("AirToGroundEngage");*
 - ♦ *HLARTI_subscribe_interaction("GroundToAirEngage");*

Registering and Deleting Objects

- Framework provides
 - Unique object IDs for registered objects
 - Single line function call for registering and deleting objects
 - String-based interface
 - Automatic exception handling
 - Automatic book-keeping of attribute and object IDs
- Federate provides
 - Object's name, ID and deletion time
 - Examples:
 - *UniqueID = HLARTI_instantiate("Ship");*
 - *HLARTI_delete(UniqueID, time);*

Updating Attributes

- Framework provides
 - Attribute Objects in C++ with operator overloading
 - Automatic sending of attributes whenever they are updated
 - Elimination of unnecessary update attribute messages
 - Automatic unit conversions, data marshalling, name translation, and exception handling, and book-keeping of attribute and object IDs
 - Federate provides
 - Initialization information
 - Example:
 - ♦ ***DOUBLE_ATTRIBUTE Speed ; // Part of an entity's state***
 - ♦ ***Speed.set_id(UniqueID); // The entity's unique ID***
 - ♦ ***Speed.set_name("Speed"); // The FOM name of the attribute***
 - ♦ ***Speed.set_knots2MPH(); // Automatic unit conversion***
 - ♦ ***Speed = 20.0; // Use as normal variable***

Sending Interactions

- **Framework provides**
 - Single line function call for the send interaction service
 - Variable length, string-based interface
 - Common packaging of interaction parameters
 - Automatic exception handling and book-keeping

- **Federate provides**
 - Interaction related information
 - Example:

*InteractionID = HLARTI_send_interaction(
 "AirToGroundEngage", InitiatorID, ReceiverID, InteractionTime,
 "launch_time", "double", LaunchTime,
 "salvo_size", "int", 20,
 etc... for other interaction parameters
 NULL);*

Receiving an Event

- **Framework provides the PostEventObject**
 - Encapsulation of all received event information for same object at same time into a single event object
 - String-based interfaces to unpack the PEOs
 - Methods to automate the reflect attribute and remove objects services and to automate attribute and object ID book-keeping
 - Standard mechanism to correctly pass received event information to the federate
- **Federate provides**
 - Its own native event processing and queue management code
 - Examples:
 - Discovering an object and reflecting its attributes
 - Removing an object
 - Receiving an interaction

Receiving an Event (Continued...)

- Example: discovering an object and reflect attributes

```
if (PEO->DISCOVER_OBJECT()) {  
    if (!strcmp(PEO->get_class_name(), "Aircraft")) {  
        AIRCRAFT *Aircraft = new AIRCRAFT();  
        NewAircraft->id = PEO->get_id();  
        // Initialize the attribute objects for the aircraft with  
        their  
        // FOM names and unique Id  
    }  
}  
PEO->ReflectAttributes();
```

Receiving an Event (Continued...)

- **Example: removing an object**

```
if (PEO->REMOVE_OBJECT()) {  
    UniqueID = PEO->get_id();  
    // find the object (however it is stored) and delete  
    it  
    PEO->RemoveObject();  
}
```

Receiving an Event (Continued...)

- **Example: receiving an interaction**

```
if (PEO->RECEIVE_INTERACTION()) {  
    int initiator = PEO->get_initiator();  
    int Receiver = PEO->get_receiver();  
    double InteractionTime = PEO->get_interaction_time();  
    if (!strcmp(PEO->get_class_name(), "AirToGroundEngage")) {  
        double Tlaunch = PEO-  
            >get_double_parameter("launch_time");  
        int SalvoSize = PEO->get_int_parameter("salvo_size");  
        // etc... for other interaction parameters  
    }  
}
```

Time Management

- Framework provides
 - Coordination of message processing and the advancement of logical time through the the main processing loop
 - Example: Framework's main processing loop

```
SimTime = NextSimTime = RTItime = 0.0;
while (SimTime < EndTime) {
    next_event_request(NextSimTime);
    while (!TimeAdvanceGrantedFlag) {
        tick() // while RTI has tick it sends the framework messages
        // messages packaged into PEOs and stored in RecEvtQ
    }
    while (PEO=RecEvtQ->RemoveItem()) HLA_PostEvent(PEO);
    NextSimTime = HLA_ProcessUpTo(RTItime); SimTime = RTItime;
    SendUpdatedAttributes();
}
```

Summary

- **Characteristics of the NSS HLA-Integration Framework**
 - *Simplified interfaces* that substitute for several key direct RTI calls
 - *Automatic* sending and receiving of updated attributes using *Attribute Objects*
 - Straight forward way to handle receiving various types of messages using *Post Event Objects*
 - *A thread of control* that also coordinates *time management*
- **Advantages of the Framework middleware approach**
 - A plan on how to cleanly integrate RTI services *while preserving standalone maintenance and operation*
 - HLA integration *with a modular object-oriented design* that is relatively quick to integrate, stable and easy to debug
 - *Reusable software or a software approach* for other DoD simulations needing to adapt to HLA